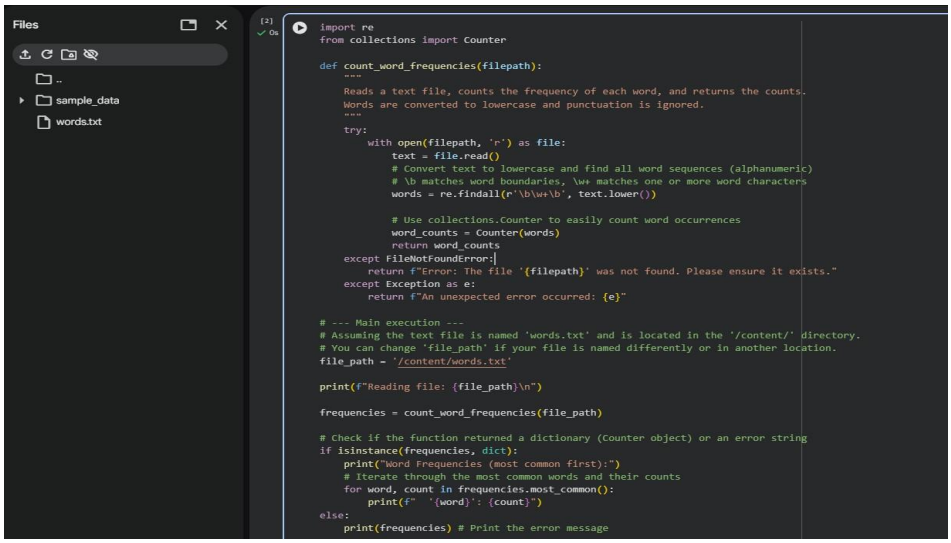
AI ASSISTED CODING

Assignment 2.3
Name: GOLI NITHYA
Batch:32

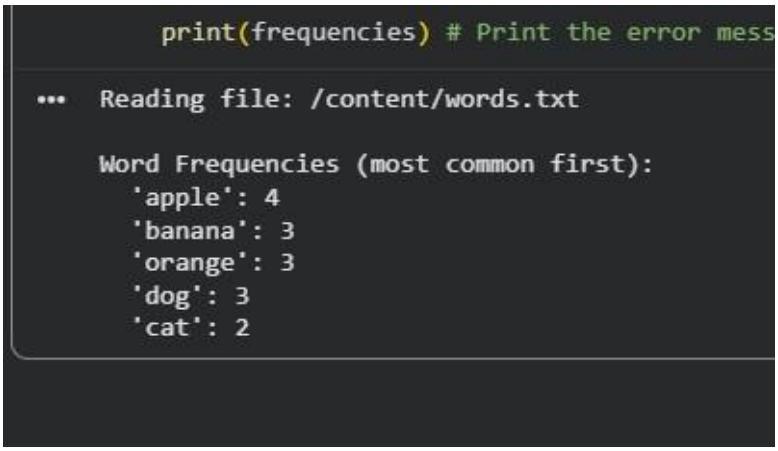
Question:

Task 1:
Word Frequency from Text File
Scenario: You are analyzing log files for keyword frequency. Task: Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code

CODE SCREENSHOT:



OUTPUT:

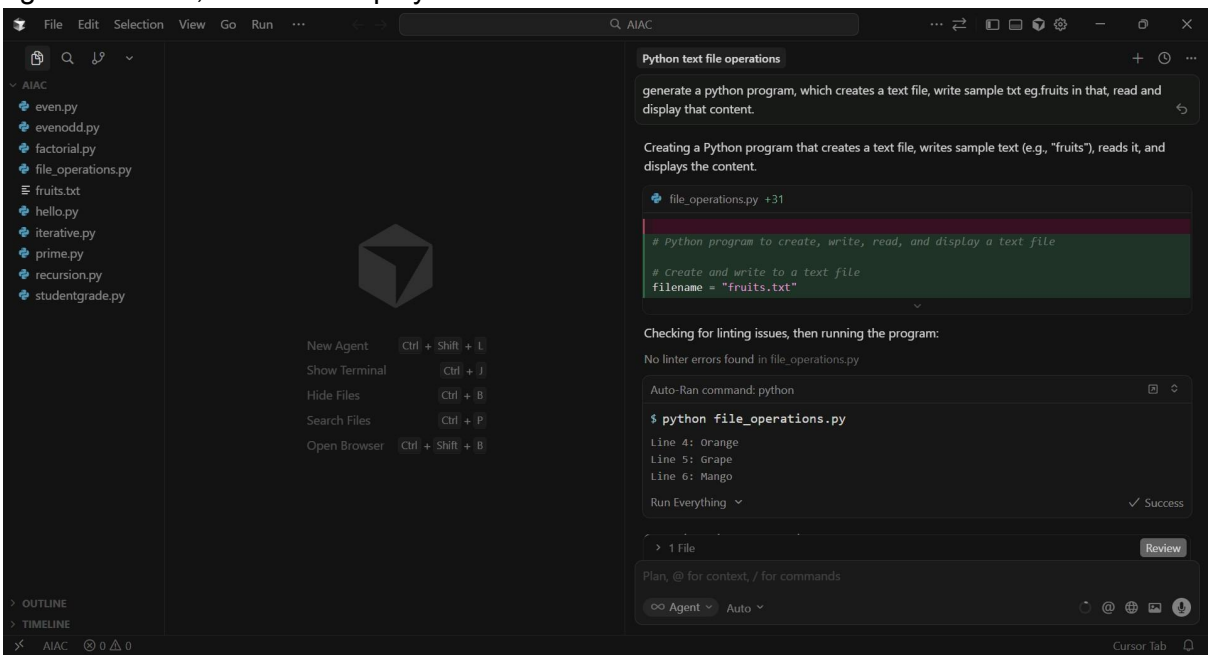


Explanation:

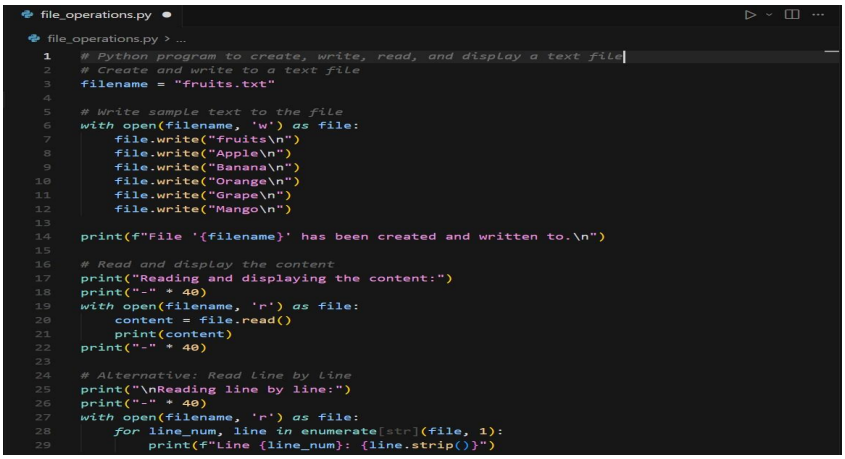
This program reads a text file and splits the content into words.
Each word is stored in a dictionary where the key is the word and the value is its count.
If a word appears again, its count is increased.
Finally, the program prints how many times each word appears in the file. This helps in analyzing text data and finding frequently used words.

Task 2:
File Operations Using Cursor AI
Scenario: You are automating basic file operations.
Task: Use Cursor AI to generate a program that:
Creates a text file
Writes sample text
Reads and displays the content

PROMT:
generate a python program, which creates a text file, write sample txt
eg.fruits in that, read and display that content.

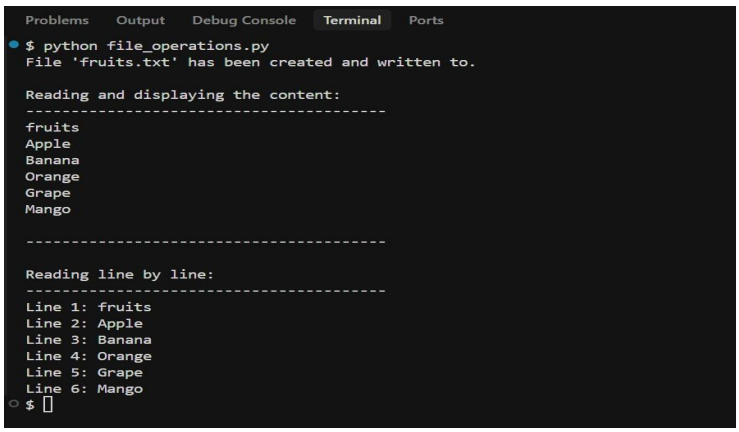


Code Screenshot:



```python
# Python program to create, write, read, and display a text file
# Create and write to a text file
filename = "fruits.txt"

# Write sample text to the file
with open(filename, 'w') as file:
    file.write("fruits\n")
    file.write("Apple\n")
    file.write("Banana\n")
    file.write("Orange\n")
    file.write("Grape\n")
    file.write("Mango\n")

print(f"File '{filename}' has been created and written to.\n")

# Read and display the content
print("Reading and displaying the content:")
print("-" * 40)
with open(filename, 'r') as file:
    content = file.read()
    print(content)
print("-" * 40)

# Alternative: Read line by line
print("\nReading line by line:")
print("-" * 40)
with open(filename, 'r') as file:
    for line_num, line in enumerate(file, 1):
        print(f"Line {line_num}: {line.strip()}")
```

OUTPUT:



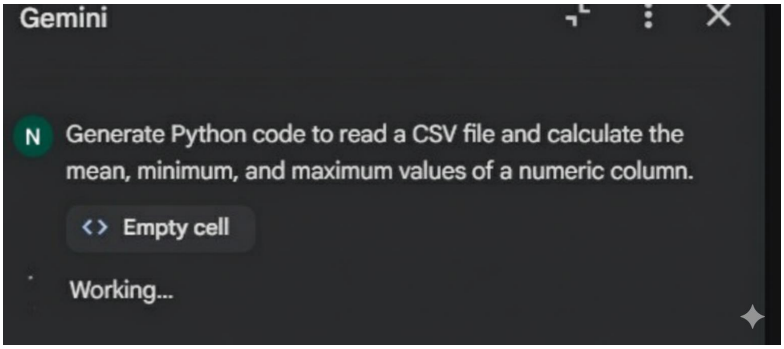
```
Problems  Output  Debug Console  Terminal  Ports
$ python file_operations.py
File 'fruits.txt' has been created and written to.

Reading and displaying the content:
----------------------------------------
fruits
Apple
Banana
Orange
Grape
Mango

----------------------------------------

Reading line by line:
----------------------------------------
Line 1: fruits
Line 2: Apple
Line 3: Banana
Line 4: Orange
Line 5: Grape
Line 6: Mango
$
```

Explanation:

This program demonstrates basic file handling in Python using Cursor.
First, a text file is created and sample text is written into it.
Then, the same file is opened in read mode and its contents are displayed on the screen.
It shows how Python can be used to create, write, and read files easily. Such operations are useful in automation and data storage tasks.

QUESTION:
Task 3: CSV Data Analysis
Scenario: You are processing structured data from a CSV file. Task: Use Gemini in Colab to read a CSV file and calculate mean, min, and max.

PROMT:
Generate Python code to read a CSV file and calculate the mean, minimum, and maximum values of a numeric column.



CODE SCREENSHOT:

```python
import pandas as pd
import io

# Uncomment the lines below and replace 'your_file.csv' with your CSV file path
try:
    df = pd.read_csv('scores.csv')
except FileNotFoundError:
    print("Error: CSV file not found. Please check the path.")
    exit()

# Specify the numeric column you want to analyze
column_name = 'score'

# Check if the column exists in the DataFrame
if column_name in df.columns:
    # Ensure the column is numeric (e.g., float or int)
    # pd.to_numeric will convert values to numeric, coercing errors to NaN
    numeric_column = pd.to_numeric(df[column_name], errors='coerce')

    # Drop rows where the numeric conversion resulted in NaN (non-numeric values)
    numeric_column = numeric_column.dropna()

    if not numeric_column.empty:
        # Calculate mean, minimum, and maximum
        mean_value = numeric_column.mean()
        min_value = numeric_column.min()
        max_value = numeric_column.max()

        print(f"Statistics for column '{column_name}':")
        print(f"  Mean: {mean_value:.2f}")
        print(f"  Minimum: {min_value:.2f}")
        print(f"  Maximum: {max_value:.2f}")
    else:
        print(f"Column '{column_name}' contains no valid numeric data after cleaning.")
else:
    print(f"Error: Column '{column_name}' not found in the CSV file. Available columns: {df.columns.tolist()}")
```

OUTPUT:

```
Statistics for column 'score':
  Mean: 82.00
  Minimum: 67.00
  Maximum: 92.00
```

EXPLANATION:

This program reads data from a CSV file using Python.
It extracts numerical values from a column and calculates the mean, minimum, and maximum.
CSV analysis is used in data processing and analytics applications.

QUESTION:

Task 4: Sorting Lists Manual vs Built-in

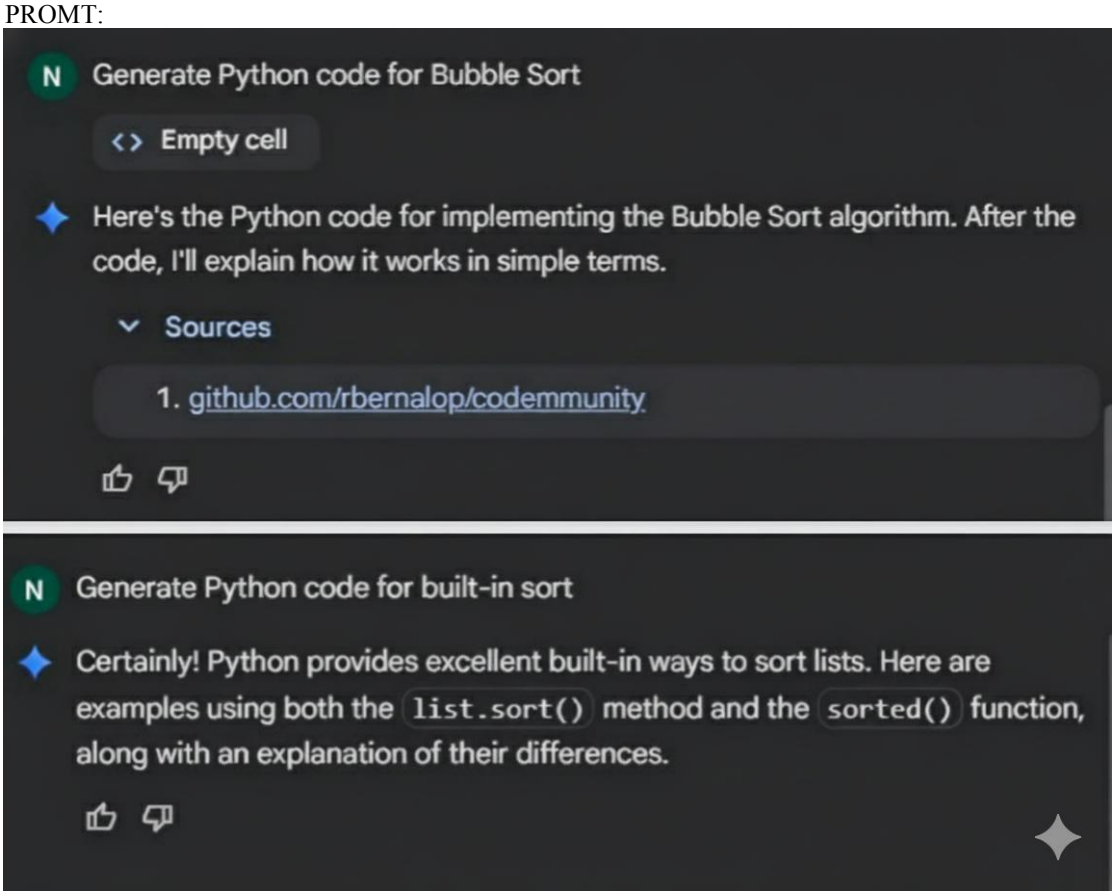
Scenario: You are reviewing algorithm choices for efficiency.

Task: Use Gemini to generate:

Bubble sort

Python's built-in sort() Compare both implementations

PROMT:



CODE SCREENSHOT:

Manual bubble Sort Code:

```
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n - i - 1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j] # Swap elements
    return arr

# --- Example Usage ---
my_list = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", my_list)

sorted_list = bubble_sort(my_list)
print("Sorted list using Bubble Sort:", sorted_list)

my_list_2 = [5, 1, 4, 2, 8]
print("\nOriginal list 2:", my_list_2)

sorted_list_2 = bubble_sort(my_list_2)
print("Sorted list 2 using Bubble Sort:", sorted_list_2)
```

Sorting using sort() function:

```
# --- Using the list.sort() method ---
# This method sorts the list in-place (modifies the original list) and returns None.

my_list_1 = [64, 34, 25, 12, 22, 11, 90]
print("Original list (list.sort()):", my_list_1)

my_list_1.sort()
print("Sorted list (list.sort()):", my_list_1)

# You can also sort in descending order
my_list_2 = ['banana', 'apple', 'cherry', 'date']
print("\nOriginal list (descending):", my_list_2)

my_list_2.sort(reverse=True)
print("Sorted list (descending):", my_list_2)
```

OUTPUT OP:

Bubble sort:

```
Original list: [64, 34, 25, 12, 22, 11, 90]
Sorted list using Bubble Sort: [11, 12, 22, 25, 34, 64, 90]

Original list 2: [5, 1, 4, 2, 8]
Sorted list 2 using Bubble Sort: [1, 2, 4, 5, 8]
```

Sort Function:

```
Original list (list.sort()): [64, 34, 25, 12, 22, 11, 90]
Sorted list (list.sort()): [11, 12, 22, 25, 34, 64, 90]

Original list (descending): ['banana', 'apple', 'cherry', 'date']
Sorted list (descending): ['date', 'cherry', 'banana', 'apple']
```

EXPLANATION:

Bubble sort is a simple sorting algorithm that repeatedly compares and swaps adjacent elements. It is easy to understand but inefficient for large data sets. Python's built-in sort function is shorter, optimized and much faster. The built-in method should be preferred in real-world applications.